CS144 Notes: Information Retrieval

General problem

- People often have "need" for information.
- How can we help people get the information that they need?
- How can we use computers to help people to answer their information need?

    - a number of ideas/approaches have been explored

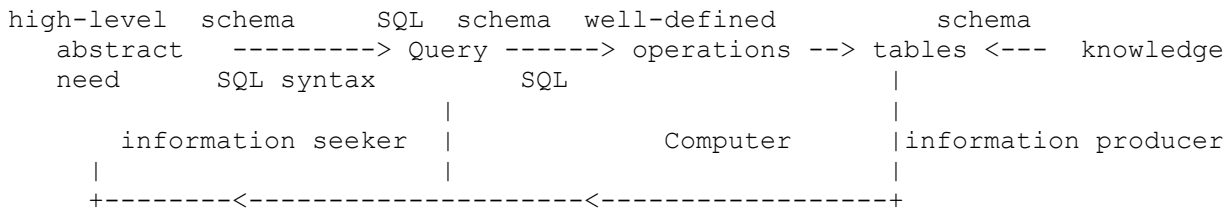
DB approach

- store the data with well-defined structure (table in RDB)
- apply exact boolean conditions on the data to filter "relevant" data only
- provide simple ways to "combine" multiple data ("joins" in RDB)

    - turns out to be extremely powerful mechanism to represent and retrieve information

    - can express a lot of interesting need or "queries"

- Q: To employ this approach, what is the overall information production/retrieval process?



- **Remarks**
    - can be extremely powerful: users can precisely describe what he wants

    - E.F. Codd got Turing award for relational model which solves
    - the problem for a very specialized domain...

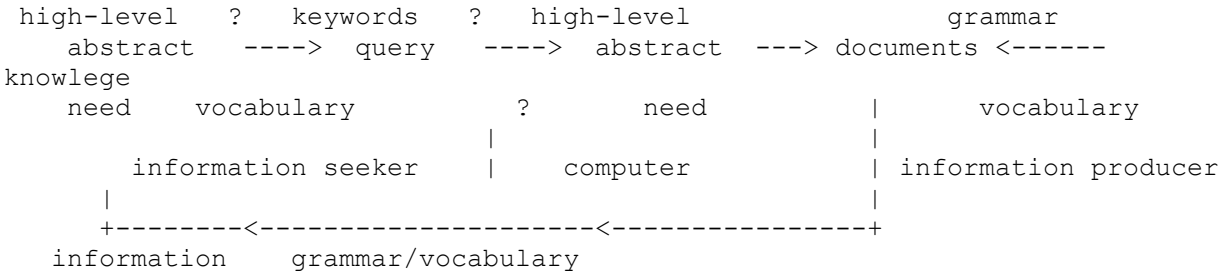
- requires significant learning both for the information producer and information seeker
    - what is the point of the expressive power when there is no data?
- The main reason for the success of this approach is because even before the solution was developed there were people who were willing to pay. businesses have been dealing with the problem forever

Old Expert system approach

- **knowledge is represented as a set of "logical predicates"**
- **our need can also be represented as logical predicates**
    - example: LivesWith(Smith, Gerald)
        - LivesWith(Gerald, Paul)
        - LivesWith(A, B) & LivesWith(B, C) -> LivesWith(A, C)
        - Q: LivesWith(Smith, Paul)?
- **This approach has had limited success**
- **Q: What is the problem of this approach?**
    - difficulty of enumerating all knowledge (hidden shared knowledge)
    - conflicting rules
        - real world
    - dependency on exact choice of predicates
        - difficulty in integrating and leveraging others' work
    - inference algorithms turns out to be intractably complex
    - Remark: Recently Bayesian network has got more popular and has had more practical impact.
        - reasoning with "probability"

IR approach

- **start with the plentiful documents**
- **the user provides "keyword queries"**
- **the system magically figures out what documents are relevant**



- **Q: What are the pros/cons of the IR approach?**
  - information producer does not have to learn a new model and language
    * potentially any literate person can be the producer
  - can use the vast amount of existing data
    * no reformatting necessary (except digitization). Google library
  - conversion process is very vague
    * how to convert need to query is not clear
    * how to convert "query" to information need is not clear
    * how to match documents to information need is not clear
  - the computer needs a "formal models" on documents, queries, and their mappings
    * boolean model, vector space model and probabilistic model
    * topic of the next lecture

Q: Any other way?

- **Remark: This is really the question of great importance.**
  - You can be the person who develops the next breakthrough
  - and move forward the humanity to the next level...

Boolean model

- user wants all documents that contain the word

- **Q: Given this model, what does the system have to do?**

- **Q: How can the system identify the documents that contain the keywords?**

  - no preprocessing or index: string matching algorithm
    - Knuth-Morris-Pratt algorithm
    - Boyce-Moore algorithm

  - index through preprocessing:
    - suffix tree
    - suffix array
    - inverted index

Precision and recall

- Boolean model is an approximation. Is it really effective in returning what people are looking for?

- Precision and recall: a measure for evaluating the effectiveness of a particular model

- Basic idea: A model is good if it returns all and only relevant documents
- R: the set of "relevant" documents to a query (determined by human)
  D: the set of documents returned by a model

- Precision: |R&D|/|D|
  - Among returned documents, what fraction is relevant?
- Recall: |R&D|/|R|
  - Among relevant documents, what fraction is returned?

Inverted index

- INVERTED INDEX: An index that has the mapping from keyword to docid

- <inverted index diagram>
- lexicon/dictionary -> Posting list (or inverted list)

- **Q: How can we use inverted index to answer "UCLA Computer Science"?**

- **Inverted index construction algorithm**
  - Q: Given document collection C, how can we build an inverted index?
  - Algorithm
    * Input: C - set of documents
    * Output: Inverted index: DIC - dictionary of inverted index
      PL(w) - posting list for word w
    * For each document d in C:
      - Extract all words in content(d) into W
      - For each w in W:
        - If w in DIC, then add w to DIC
        - Append id(d) to PL(w)

- **Size estimation of inverted index**
  - Example: 100M docs, 10KB/doc, 1000 unique words/doc, 10 bytes/word
    - 4 byte/docid
    * Q: Document collection size?

* Q: Inverted index size?
    - Q: total # docids in posting lists?
    - Q: # words in dictionary?
    - Q: Between dictionary and posting lists, which one is larger?

- Remark: # of unique words grows with # of docs at the rate of O(n^a)
    * where a is between 0.5 to 1
- Rule of thumb: inverted index size is roughly equivalent to corpus size

- **Q: How can we construct inverted index if the size is larger than memory?**

Keyword queries for RDB

- **Example: Online forum**
    - Users(uid, name, email, password)
    - Postings(pid, threadid, uid, time, title, body)
        * Q: All postings from uid = 'expert01' in SQL?

- Q: How can DBMS answer it? Any efficient way?

* Q: All postings with keywords 'web' and 'application' in body in SQL?

- Q: How can DBMS answer it? Any efficient way?

* Q: All postings with keywords 'web' and 'application' in body in SQL?

- Q: How can DBMS answer it? Any efficient way?

* Remark:
  - SQL is not the best way to express keyword queries
  - Standard DBMS index is not suitable for handling keyword queries
  - Inverted index on some attributes are helpful

Vector model

- **Main problem of boolean model**
  - When document collection is large, there may be too many documents matching a set of keywords. Are they equally good? How can we help users to identify that are more likely to be relevant?
- **Matrix representation of the Boolean model**

- Q: But, "Princeton" and "University", are they really equally important?

- **Basic idea of vector model**
  - instead of 0 and 1 for terms, give real-valued weight for each term
    - depending on how important the term is for the document

- **Q: How do we know whether a term is "important" for a document?**

- **Basic idea of tf-idf weighting (or tf-idf vector)**
  - a term t is important for document D
    - if t appears many times in D or
    - if t is a rare term
  - tf: term frequency: # occurrence of t in D
  - idf: inverse document frequency: # documents containing t
    - more precisely, N: total # documents, n: # documents with t,
      - f: # occurrence of t in D
  - tf.idf = f x log (N/n)
    - weight is high if the term is rare, but appears often in the document

- **Cosine similarity**
  - represent the query using the same tf-idf vector
    - similarly to documents, a query term is "important" if it is a rare term
  - take inner product of the two vectors to compute "similarity" of documents to query
    - cosine similarity will be high only if query and documents share many rare terms
      - assume idf(Princeton)=1,
        idf(university)=idf(car)=idf(racing)=idf(admission)=0.1

- Q = (Princeton, university)

* D1 = (car, racing). D1.Q = ?

* D2 = (Princeton, admission) D2.Q = ?

* D3 = (university, admission) D3.Q = ?

- sort documents by the cosine similarity value

- **Q: How can we compute cosine similarity quickly?**

- Q: What additional information should we include in inverted index?

- **Q: Under the vector model, does precision/recall still make sense?**

- Remark: precision @ 10. more useful when the # docs is large

Topic-based Indexing

- Main motivation
  - vector model still does not return a page on cars when the query is automobiles
  - we should index documents based on the "concepts" mentioned in the document not by the terms
    1. hopefully fewer concepts than terms. so smaller document vector
    2. returns document with the term "car" when the query is automobile

- the desired mapping:

    document --> concept --> associated terms

    <example (concept -> term) and (doc -> concept) mappings>

```
          car  automobile  movie  theater
car    (  1,         0.9,     0,        0)
movie  (  0,           0,     1,      0.8)

----
         car, movie
doc1  (   0,     1)
doc2  (   1,     0)
doc3  ( 0.8,   0.2)
 ...
----
doc1 = (0, 0, 1, 0.8)          = 0 (1, 0.9, 0, 0) + 1 (0, 0, 1, 0.8)
doc2 = (1, 0.9, 0, 0)          = 1 (1, 0.9, 0, 0) + 0 (0, 0, 1, 0.8)
doc3 = (0.8, 0.72, 0.2, 0.16) = 0.8 (1, 0.9, 0, 0) + 0.2 (0, 0, 1, 0.8)
```

    Remark: we get terms in a document through this indirect mapping through concepts

    * In general,
    * (- dt1 -) = dc11 (- ct1 -) + dc12 (- ct2 -) + ... dc1k (- ctk -)

- MATRIX INTERPRETATION: we want to get document-term matrix through concept

```
DC =           CT =
                term                  concept          term
           | - dt1 -   |           | - dc1 -  |    | - ct1 -  |
 DT = doc  |    ...    |  ->  doc  |  ...     | X  |   ...    |
           | - dtn -   |           | - dcn -  |    | - ctk -  |
```

    * (- dt1 -) = dc11 (- ct1 -) + dc12 (- ct2 -) + ... dc1k (- ctk -)

- dti: document-term vector
- dci: document-concept vector
- cti: concept-term vector

- DT: document-term matrix (n x m)
- DC: document-concept matrix. (n x k)
- CT: concept-term matrix (k x t)

        * DT = DC CT should be close to the original T matrix

        * k is assumed to be much smaller than m

    - Q: How do we extract "concepts" from "terms"?
      How can we do this "decomposition"?

- **Latent semantic indexing**

    - What we do is to approximate the (nxm) matrix

    - Do SVD on document-term matrix and keep the top-k singular values

        * M = D S T

    - represent documents by the (doc-concept) matrix.

        * compute the similarity of documents based on the doc-concept matrix

- **Probabilistic topic model (Latent Dirichlet Analysis)**

    - Probabilistic document generation model and probabilistic topic inference

<u>Link-based ranking</u>

- **Main issue:**

    - Given a query q, vector model returns pages that contain q many times, which may not be what we want.

    - Q: For query "American Airlines", what should be the result?

    - Q: Under a vector model, will "American Airlines" homepage be ranked high?

        * Remark: Crash reports for "American Airlines"

    - Q: How can search engine know what is the "expected" for a query?

    - Remark: Difficulty of knowing "what users expect"

- Q: What can we do?

- Remark:

    - "common" knowledge is well-known
        - Amazon, AA, UCLA home pages

    - consider your home page and the link to UCLA homepage
        - many links to well-known pages
        - anchor text is often a succinct summary of the page

- **second-generation search engines**

    - Idea 1: Give much higher weights to the anchor text

    - Idea 2: Rank a page high if the page has many incoming links

    - Q: How does this idea change the document-term matrix and the ranking function?

- **Link-based ranking**
    - First idea: Count # incoming links to a page and use it for ranking

        - Q: Any problem?

    - Second idea: Not every page is equal. Give a high weight to a link from an "important" page like Yahoo.

        - Q: How much weight do we give to each link?
          Assuming P(pi) is the importance of page pi, what should the formula of P(pi) be?

- **PageRank**
  - P(pi) = 1/c1 P(p1) + ... + 1/ck P(pk)
    * p1,...,pk: pages that have a link to pi
    * ci: # of links in pi
    * Q: Recursion? Can we really solve the above recursive formula?
  - Matrix representation of PageRank formula
    * P = (P(p1), P(p2), ..., P(pn)): PageRank vector
    * M = | mij | where mij = 1/cj if pj has a link to pi: Web graph matrix
    * <show Amazon, Google, Microsoft example and construct the matrix>
    * P = M P
      - The common way of computing P is through recursion
      - Q: How can we compute it through recursion?
        - two arrays for old values and new PageRank values
        - disk-based link data
      - under typical settings (d = 0.85)
        - after 10 iterations 20% error.
        - after 20 iterations 4% error
        - after 30 iterations 0.8% error

      <For the example, principal eigenvector is (n, m, a) = (6/5, 3/5, 6/5)>
  - Random-surfer model
    * Consider a user who does a random browsing of the Web
      - starts from a random page
      - clicks on a link from the page randomly
    * Q: What is the probability of a user arriving at page pi?
      - P(pi): probability to arrive at page pi.
      - Q: What is P(pi)?

- Q: How does the user arrive pi?
    - by following a link to page pi
- Q: Then what is P(pi)?
    - sum up the probability from every page that links to pi
    - P(pi) = 1/c1 P(p1) + ... + 1/ck P(pk)

* P(pi) is the probability of for a random web surfer to arrive at pi.

- Problems of PageRank and damping factor
    * Two problems of PageRank
        1. Dangling page: Values dissipates through pages without any links
        2. Crawler trap: Pages that just point to each other.
           All values may accumulate on those page
- <show the dangling page and crawler-trap examples
  and show how the values change through iteration>

  <for dangling page example, (n,m,a)->(0,0,0)>

  <for crawler-trap example, (n,m,a)->(0,3,0)>

    * Handling dangling page
        - Assume the user jumps to a random page if no links
    * Handling crawler trap
        - Damping factor: For every page, there is a probability (1-d) that the user jumps to a random page
            - When the user gets bored, he jumps to a random page with probability 1-d
    * Q: How does the equation change?

- P(pi) = d [1/c1 P(p1) + ... + 1/ck P(pk)] + (1-d)/n

<compute the PageRank values with these changes>

<with d=4/5, (n,m,a) -> (7/11,21/11,5/11)>

- **Q: Assume some linear combination of cosine similarity and PageRank for final ranking. How can we compute the ranking efficiently?**

  - Remark:
    * PageRank may be stored as part of inverted index or separately.
    * If stored separately, random access to PageRank values should be efficient
      - in-memory storage necessary

IR engine

- **Supports efficient keyword queries on document collection**
  - Given a set of documents, builds inverted index
  - Given a query, quickly return matching documents
  - Results are ranked (typically by tf-idf metric)
- **Lucene**
  - Java based open-source IR library that provides
    * inverted index
    * query processing
  - Document ranking is based on tf-idf metric (discussed later)
  - Can be used together with RDB to support efficient keyword queries